一旦，我们建立好了tcp连接之后，我们就可以把得到的fd当作文件描述符来使用。  
由此网络程序里最基本的函数就是read和 **write**函数了。  
ssize\_t **write**(int fd, const void\*buf,size\_t nbytes);  
**write**函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数.失败时返回-1. 并设置errno变量. 在网络程序中,当我们向套接字文件描述符写时有两可能.  
1)**write**的返回值大于0,表示写了部分或者是全部的数据. 这样我们用一个while循环来不停的写入，但是循环过程中的buf参数和nbyte参数得由我们来更新。也就是说，网络写函数是不负责将全部数据写完之后在返回的。  
2)返回的值小于0,此时出现了错误.我们要根据错误类型来处理.  
如果错误为EINTR表示在写的时候出现了中断错误.  
如果为EPIPE表示网络连接出现了问题(对方已经关闭了连接).  
为了处理以上的情况,我们自己编写一个写函数来处理这几种情况.  
  
int my\_**write**(int fd,void \*buffer,int length)  
{  
int bytes\_left;  
int written\_bytes;  
char \*ptr;  
  
ptr=buffer;  
bytes\_left=length;  
while(bytes\_left>0)  
{  
          
         written\_bytes=**write**(fd,ptr,bytes\_left);  
         if(written\_bytes<=0)  
         {         
                 if(errno==EINTR)  
                         written\_bytes=0;  
                 else               
                         return(-1);  
         }  
         bytes\_left-=written\_bytes;  
         ptr+=written\_bytes;       
}  
return(0);  
}  
  
读函数read  
ssize\_t read(int fd,void \*buf,size\_t nbyte)  
read函数是负责从fd中读取内容.当读成功 时,read返回实际所读的字节数,如果返回的值是0 表示已经读到文件的结束了,小于0表示出现了错误.如果错误为EINTR说明读是由中断引起 的, 如果是ECONNREST表示网络连接出了问题. 和上面一样,我们也写一个自己的读函数.  
  
int my\_read(int fd,void \*buffer,int length)  
{  
int bytes\_left;  
int bytes\_read;  
char \*ptr;  
    
bytes\_left=length;  
while(bytes\_left>0)  
{  
    bytes\_read=read(fd,ptr,bytes\_read);  
    if(bytes\_read<0)  
    {  
      if(errno==EINTR)  
         bytes\_read=0;  
      else  
         return(-1);  
    }  
    else if(bytes\_read==0)  
        break;  
     bytes\_left-=bytes\_read;  
     ptr+=bytes\_read;  
}  
return(length-bytes\_left);  
}  
  
数据的传递  
有了上面的两个函数,我们就可以向客户端或者是服务端传递数据了.比如我们要传递一个结构.可以使用如下方式  
  
  
  
struct my\_struct my\_struct\_client;  
**write**(fd,(void \*)&my\_struct\_client,sizeof(struct my\_struct);  
  
  
char buffer[sizeof(struct my\_struct)];  
struct \*my\_struct\_server;  
read(fd,(void \*)buffer,sizeof(struct my\_struct));  
my\_struct\_server=(struct my\_struct \*)buffer;      
  
在网络上传递数据时我们一般都是把数据转化为char类型的数据传递.接收的时候也是一样的注意的是我们没有必要在网络上传递指针(因为传递指针是没有任何意义的,我们必须传递指针所指向的内容)  
  
  
6.1 recv和 **send**  
recv和**send**函数提供了和read和**write**差不多的功能.不过它们提供了第四个参数来控制读写操作.  
  
int recv(int sockfd,void \*buf,int len,int flags)  
int **send**(int sockfd,void \*buf,int len,int flags)  
  
前面的三个参数和read, **write**一样,第四个参数可以是0或者是以下的组合  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
| MSG\_DONTROUTE | 不查找表 |  
| MSG\_OOB | 接受或者发送带外数据 |  
| MSG\_PEEK | 查看数据,并不从系统缓冲区移走数据 |  
| MSG\_WAITALL | 等待所有数据 |  
|--------------------------------------------------------------|  
  
MSG\_DONTROUTE:是 **send**函数使用的标志.这个标志告诉IP.目的主机在本地网络上面,没有必要查找表.这个标志一般用网络诊断和路由程序里面.  
MSG\_OOB:表示可以接收和发送带外的数据.关于带外数据我们以后会解释的.  
  
MSG\_PEEK:是recv函数的使用标志, 表示只是从系统缓冲区中读取内容,而不清除系统缓冲区的内容.这样下次读的时候,仍然是一样的内容.一般在有多个进程读写数据时可以使用这个标志.  
  
MSG\_WAITALL 是recv函数的使用标志,表示等到所有的信息到达时才返回.使用这个标志的时候recv回一直阻塞,直到指定的条件满足,或者是发生了错误. 1)当读到了指定的字节时,函数正常返回.返回值等于len 2)当读到了文件的结尾时,函数正常返回.返回值小于len 3)当操作发生错误时,返回-1,且设置错误为相应的错误号(errno)

MSG\_NOSIGNAL is a flag used by **send**() in some implementations of the Berkeley sockets [API](http://www.wlug.org.nz/API).

This flag requests that the implementation does not to **send** a [SIGPIPE](http://www.wlug.org.nz/SIGPIPE) signal on errors on stream oriented sockets when the other end breaks the connection. The [EPIPE](http://www.wlug.org.nz/EPIPE) error is still returned as normal.

Though it is in some Berkely sockets APIs (notably Linux) it does not exist in what some refer to as the reference implementation, [FreeBSD](http://www.wlug.org.nz/FreeBSD), which instead uses a socket option SO\_NOSIGPIPE[?](http://www.wlug.org.nz/SO_NOSIGPIPE?action=create). 对于服务器端，我们可以使用这个标志。目的是不让其发送SIG\_PIPE信号，导致程序退出。

如果flags为0,则和read,**write**一样的操作.还有其它的几个选项,不过我们实际上用的很少,可以查看 Linux Programmer's Manual得到详细解释